Cardinality of Equivalence Relations

Lukas Bulwahn

October 27, 2022

Abstract

This entry provides formulae for counting the number of equivalence relations and partial equivalence relations over a finite carrier set with given cardinality.

To count the number of equivalence relations, we provide bijections between equivalence relations and set partitions [4], and then transfer the main results of the two AFP entries, Cardinality of Set Partitions [1] and Spivey’s Generalized Recurrence for Bell Numbers [2], to theorems on equivalence relations. To count the number of partial equivalence relations, we observe that counting partial equivalence relations over a set $A$ is equivalent to counting all equivalence relations over all subsets of the set $A$. From this observation and the results on equivalence relations, we show that the cardinality of partial equivalence relations over a finite set of cardinality $n$ is equal to the $n + 1$-th Bell number [3].

Contents

1 Cardinality of Equivalence Relations 2
   1.1 Bijection between Equivalence Relations and Set Partitions  2
      1.1.1 Possibly Interesting Theorem for HOL.Equiv-Relations  2
      1.1.2 Dedicated Facts for Bijection Proof  2
      1.1.3 Bijection Proof  3
   1.2 Finiteness of Equivalence Relations  4
   1.3 Cardinality of Equivalence Relations  4

2 Cardinality of Partial Equivalence Relations 5
   2.1 Definition of Partial Equivalence Relation  5
   2.2 Construction of all Partial Equivalence Relations for a Given Set  6
   2.3 Injectivity of the Set Construction  7
   2.4 Cardinality Theorem of Partial Equivalence Relations  8
1 Cardinality of Equivalence Relations

theory Card-Equiv-Relations
imports
  Card-Partitions, Card-Partitions
  Bell-Numbers-Spivey.Bell-Numbers
begin

1.1 Bijection between Equivalence Relations and Set Partitions

1.1.1 Possibly Interesting Theorem for HOL.Equiv-Relations

This theorem was historically useful in this theory, but is now after some proof refactoring not needed here anymore. Possibly it is an interesting fact about equivalence relations, though.

lemma equiv-quotient-eq-quotient-on-UNIV:
assumes equiv A R
shows A // R = (UNIV // R) − {{}}
proof
show UNIV // R − {{}} ⊆ A // R
proof
fix X
assume X ∈ UNIV // R − {{}}
from this obtain x where X = R "{x}" and X ≠ {}
  by (auto elim!: quotientE)
from this have x ∈ A
    using ⟨equiv A R⟩ equiv-class-eq-iff by fastforce
from this ⟨X = R "{x}"⟩ show X ∈ A // R
    by (auto intro!: quotientI)
qed
next
show A // R ⊆ UNIV // R − {{}}
proof
fix X
assume X ∈ A // R
from this have X ≠ {}
  using ⟨equiv A R⟩ in-quotient-imp-non-empty by auto
moreover from ⟨X ∈ A // R⟩ have X ∈ UNIV // R
  by (metis UNIV-I assms proj-Eps proj-preserves)
ultimately show X ∈ UNIV // R − {{}} by simp
qed

1.1.2 Dedicated Facts for Bijection Proof

lemma equiv-relation-of-partition-of:
assumes equiv A R
shows \{(x, y). \exists X ∈ A // R. x ∈ X ∧ y ∈ X\} = R
proof
show \{(x, y). \exists X \in A // R. x \in X \land y \in X\} \subseteq R
proof
fix xy
assume xy \in \{(x, y). \exists X \in A // R. x \in X \land y \in X\}

next
show R \subseteq \{(x, y). \exists X \in A // R. x \in X \land y \in X\}
proof
fix xy
assume xy \in R
obtain x y where xy = (x, y) by fastforce

qed

1.1.3 Bijection Proof

lemma bij-betw-partition-of:
bij-betw \((\lambda R. A // R) \{R. {\text{equiv A R}} \ {\text{partition-on A P}}\}\)
proof (rule bij-betw-byWitness[where \(f' = \lambda P. \{(x, y). \exists X \in P. x \in X \land y \in X\}\]])
show \(\forall R \in \{R. {\text{equiv A R}} \ {\text{partition-on A P}}\}\).
\{(x, y). \exists X \in A // R. x \in X \land y \in X\} = R
by (simp add: equiv-relation-of-partition-of)
show \(\forall P \in \{P. {\text{partition-on A P}}\}. A // \{\{x, y\}. \exists X \in P. x \in X \land y \in X\} = P\)
by (simp add: partition-on-quotient)
show \(\{R. A // R\} \ {\text{equiv A P}} \subseteq \{P. {\text{partition-on A P}}\}\)
using partition-on-quotient by auto
show \((\lambda P. \{(x, y). \exists X \in P. x \in X \land y \in X\}) \ {\text{equiv A P}} \subseteq \{R. {\text{partition-on A P}}\}\)
by (simp add: equiv-relation-of-partition-of)
equiv A R
  using equiv-partition-on by auto
qed

lemma bij-betw-partition-of-equiv-with-k-classes:
  bij-betw (λR. A // R) {R. equiv A R ∧ card (A // R) = k} {P. partition-on A P ∧ card P = k}
proof (rule bij-betw-byWitness[where f′=λP. {(x, y). ∃X∈P. x ∈ X ∧ y ∈ X}]]
  show ∀R∈{R. equiv A R ∧ card (A // R) = k}. ∃X∈A // R. x ∈ X ∧ y ∈ X}
    by (auto simp add: equiv-relation-of-partition-of)
  show ∀P∈{P. partition-on A P ∧ card P = k}. A // {(x, y). ∃X∈P. x ∈ X ∧ y ∈ X} = P
    by (auto simp add: partition-on-eq-quotient)
  using partition-on-quotient by auto
  show (λP. {(x, y). ∃X∈P. x ∈ X ∧ y ∈ X}) ◦ {P. partition-on A P ∧ card P = k} ⊆ {R. equiv A R ∧ card (A // R) = k}
    by (auto simp add: partition-on-quotient)
  using equiv-partition-on by (auto simp add: partition-on-quotient)
qed

1.2 Finiteness of Equivalence Relations

lemma finite-equiv:
  assumes finite A
  shows finite {R. equiv A R}
proof
  have bij-betw (λR. A // R) {R. equiv A R} {P. partition-on A P}
    by (rule bij-betw-partition-of)
  from this show finite {R. equiv A R}
    using finite A; finitely-many-partition-on by (simp add: bij-betw-finite)
qed

1.3 Cardinality of Equivalence Relations

theorem card-equiv-rel-eq-card-partitions:
  card {R. equiv A R} = card {P. partition-on A P}
proof
  have bij-betw (λR. A // R) {R. equiv A R} {P. partition-on A P}
    by (rule bij-betw-partition-of)
  from this show card {R. equiv A R} = card {P. partition-on A P}
    by (rule bij-betw-same-card)
qed

corollary card-equiv-rel-eq-Bell:
  assumes finite A
  shows card {R. equiv A R} = Bell (card A)
using assms Bell-altdef card-equiv-rel-eq-card-partitions by force
corollary card-equiv-rel-eq-sum-Stirling:

assumes finite A

shows card \{ R. equiv A R \} = sum (Stirling (card A)) \{..card A\}

using assms card-equiv-rel-eq-Bell Bell-Stirling-eq by simp

theorem card-equiv-k-classes-eq-card-partitions-k-parts:

{ R. equiv A R } \land (card (A // R) = k) = card \{ P. partition-on A P \land card P = k \}

proof

have bij_betw (\lambda R. A // R \} { R. equiv A R \land card (A // R) = k \} \{ P. partition-on A P \land card P = k \}

by (rule bij_betw-partition-of-equiv-with-k-classes)

from this show card \{ R. equiv A R \land card (A // R) = k \} = card \{ P. partition-on A P \land card P = k \}

by (rule bij_betw-same-card)

qed

corollary

assumes finite A

shows card \{ R. equiv A R \land card (A // R) = k \} = Stirling (card A) k

using card-equiv-k-classes-eq-card-partitions-k-parts

\{card-partition-on[\text{OF} \text{finite A}\} by metis

end

2 Cardinality of Partial Equivalence Relations

theory Card-Partial-Equiv-Relations

imports

Card-Equiv-Relations

begin

2.1 Definition of Partial Equivalence Relation

definition partial-equiv :: 'a set \Rightarrow ('a \times 'a) set \Rightarrow bool

where

partial-equiv A R = (R \subseteq A \times A \land sym R \land trans R)

lemma partial-equiv:

assumes R \subseteq A \times A sym R trans R

shows partial-equiv A R

using assms unfolding partial-equiv-def by auto

lemma partial-equiv iff:

shows partial-equiv A R \iff (\exists A' \subseteq A. equiv A' R)

proof

assume partial-equiv A R

from partial-equiv A R have R " A \subseteq A

unfolding partial-equiv-def by blast

5
moreover have \texttt{equiv} \((R \preceq A) R\)

\begin{verbatim}
proof (rule equivI)
  from \texttt{partial-equiv A R} show \texttt{sym R}
  unfolding \texttt{partial-equiv-def} by blast
  from \texttt{partial-equiv A R} show \texttt{trans R}
  unfolding \texttt{partial-equiv-def} by blast
show \texttt{refl-on} \((R \preceq A) R\)
proof (rule refl-onI)
  show \texttt{R} \subseteq \texttt{R \preceq A \times R \preceq A}
  proof
    fix \texttt{p}
    assume \texttt{p} \in \texttt{R}
    obtain \texttt{x y} where \texttt{p} = \texttt{(x, y)} by fastforce
    moreover have \texttt{x} \in \texttt{R \preceq A}
      using \texttt{p} \in \texttt{R} \texttt{p} = \texttt{(x, y)} \texttt{partial-equiv A R}
      partial-equiv-def sym-def by fastforce
    moreover have \texttt{y} \in \texttt{R \preceq A}
      using \texttt{p} \in \texttt{R} \texttt{p} = \texttt{(x, y)} \texttt{R \preceq A} \subseteq \texttt{A} \texttt{x} \in \texttt{R \preceq A}
      by blast
    ultimately show \texttt{p} \in \texttt{R \preceq A \times R \preceq A} by auto
    qed
  next
    fix \texttt{y}
    assume \texttt{y} \in \texttt{R \preceq A}
    from \texttt{this} obtain \texttt{x} where \texttt{(x, y)} \in \texttt{R} by auto
    from \texttt{(x, y)} \in \texttt{R} have \texttt{(y, x)} \in \texttt{R}
      using \texttt{sym R} by (meson symE)
    from \texttt{(x, y)} \in \texttt{R} \texttt{(y, x)} \in \texttt{R} show \texttt{(y, y)} \in \texttt{R}
      using \texttt{trans R} by (meson transE)
    qed
    qed
ultimately show \exists \texttt{A' \subseteq A}. \texttt{equiv A' R} by blast
next
assume \exists \texttt{A' \subseteq A}. \texttt{equiv A' R}
from \texttt{this} obtain \texttt{A'} where \texttt{A' \subseteq A} and \texttt{equiv A' R} by blast
show \texttt{partial-equiv A R}
proof (rule partial-equivI)
  from \texttt{equiv A' R} \texttt{A' \subseteq A} show \texttt{R \subseteq A \times A}
    using equiv-class-eq-iff by fastforce
  from \texttt{equiv A' R} show \texttt{sym R}
    using equivE by blast
  from \texttt{equiv A' R} show \texttt{trans R}
    using equivE by blast
  qed
  qed
\end{verbatim}

\textbf{2.2 Construction of all Partial Equivalence Relations for a Given Set}

\begin{verbatim}
definition all-partial-equivs-on :: \'a set \Rightarrow ((\'a \times \'a) set) set
\end{verbatim}
where

all-partial-equis-on A =
do {
  k ← {0..card A};
  A' ← {A'. A' ⊆ A ∧ card A' = k};
  {R. equiv A' R}
}

lemma all-partial-equis-on:
assumes finite A
shows all-partial-equis-on A = {R. partial-equiv A R}
proof
show all-partial-equis-on A ⊆ {R. partial-equiv A R}
proof
  fix R
  assume R ∈ all-partial-equis-on A
  from this obtain A' where A' ⊆ A and equiv A' R
  unfolding all-partial-equis-on-def by auto
  from this have partial-equiv A R
  using partial-equiv-iff by blast
  from this show R ∈ {R. partial-equiv A R} ..
qed
next
  show {R. partial-equiv A R} ⊆ all-partial-equis-on A
  proof
    fix R
    assume R ∈ {R. partial-equiv A R}
    from this obtain A' where A' ⊆ A and equiv A' R
    using partial-equiv-iff by (metis mem-Collect-eq)
    moreover have card A' ∈ {0..card A}
    using A' ⊆ A by (simp add: card-mono)
    ultimately show ?thesis
      using mem-Sigma-iff subset-antisym subset-eq by blast
  qed
qed

2.3 Injectivity of the Set Construction

lemma equiv-inject:
  assumes equiv A R equiv B R
  shows A = B
proof –
  from assms have R ⊆ A × A R ⊆ B × B by (simp add: equiv-type)+
  moreover from assms have ∀x∈A. (x, x) ∈ R ∀x∈B. (x, x) ∈ R
    by (simp add: eq-equiv-class)+
  ultimately show ?thesis
    using mem-Sigma-iff subset-antisym subset-eq by blast
qed
lemma injectivity:
assumes \((A' \subseteq A \land \text{card } A' = k) \land (A'' \subseteq A \land \text{card } A'' = k')\)
assumes \(\text{equiv } A' R \land \text{equiv } A'' R'\)
shows \(A' = A''\) \(k = k'\)
proof -
  from \(R = R'\) assms(2) show \(A' = A''\)
  using equiv-inject by fast
  from this assms(1) show \(k = k'\) by simp
qed

2.4 Cardinality Theorem of Partial Equivalence Relations

theorem card-partial-equiv:
assumes \(\text{finite } A\)
shows \(\text{card } \{R. \text{partial-equiv } A R\} = \text{Bell (card } A + 1\)\)
proof -
  let \(?expr = do \{\)
    \(k \leftarrow \{0..\text{card } A\};\)
    \(A' \leftarrow \{A'. \ A' \subseteq A \land \text{card } A' = k\};\)
    \(\{R. \text{equiv } A' R\}\)
  \}
  have \(\text{card } \{R. \text{partial-equiv } A R\} = \text{card } (\text{all-partial-equivs-on } A)\)
    using \(\text{finite } A\) by (simp add: all-partial-equivs-on)
  also have \(\text{card } (\text{all-partial-equivs-on } A) = \text{card } ?expr\)
    unfolding all-partial-equivs-on-def ..
  also have \(\text{card } ?expr = (\sum k = 0..\text{card } A. (\text{card } A \text{ choose } k) \ast \text{Bell } k)\)
    proof -
      let \(?S \gg ?comp = ?expr\)
      \{
        \fix k
        assume \(k: k \in \{0..\text{card } A\}\)
        let \(?expr = ?comp k\)
        let \(?S \gg \text{finite } ?S \text{ using } \text{finite } A\) by simp
        moreover \{
          \fix A'
          assume \(A': A' \in \{A'. \ A' \subseteq A \land \text{card } A' = k\}\)
          from this have \(A' \subseteq A \text{ and card } A' = k\) by auto
          let \(?expr = ?comp A'\)
          have finite \(?expr\)
            using \(\text{finite } A\) \(\text{finite } A' \subseteq A\) by (simp add: finite-subset)
          have \(\text{card } ?expr = \text{Bell } k\)
            using \(\text{finite } A\) \(\text{finite } A' \subseteq A\) \(\text{card } A' = k\)
            by (simp add: card-equiv-rel-eq-Bell)
          moreover have finite \(?expr\)
            using \(\text{finite } A'\) by (simp add: finite-equiv)
          ultimately have finite \(?expr\) \text{ and card } ?expr = \text{Bell } k\) by blast
      \}
  \}
moreover have disjoint-family-on ?comp ?S
  by (injectivity-solver rule: injectivity(1))
moreover have card ?S = card A choose k
  using finite A; by (simp add: n-subsets)
ultimately have card ?expr = (card A choose k) \* Bell k (is - = ?formula)
  by (subst card-bind-constant) auto
moreover have finite ?expr
  using finite ?S inner by (auto intro!: finite-bind)
ultimately have finite ?expr \& card ?expr = ?formula by blast
}
moreover have finite ?S by simp
moreover have disjoint-family-on ?comp ?S
  by (injectivity-solver rule: injectivity(2))
ultimately show card ?expr = (∑k = 0..card A. (card A choose k) \* Bell k)
  by (subst card-bind) auto
qed
also have \ldots = (∑k≤card A. (card A choose k) \* Bell k)
  by (auto intro: sum.cong)
also have \ldots = Bell (card A + 1)
  using Bell-recursive-eq by simp
finally show ?thesis.
qed

end

References


